# Data Structures and Algorithms

## - Dynamic Hashing -

School of Electrical Engineering
Korea University

# Dynamic External Hashing

- Extendible Hashing
  - use an access structure in addition to the file
  - based on the result of the hash function to the search field
  - similar to index but based on the search field.
- Linear Hashing
  - do not need any access structure
  - based on a sequence of hash functions

# Extendible Hashing(1)

- A directory can be stored on disk, and it expands or shrinks dynamically. Directory entries point to the disk blocks that contain the stored records.
- A directory of $2^d$ bucket addresses, where *d* is called the global depth of the directory.
- The first *d* bits of a hash value as an index into the directory.
- Several directory locations with the same first *d'* (local depth) bit for their hash values may contain the same bucket address if all the records that hash to these locations fit in a single bucket.

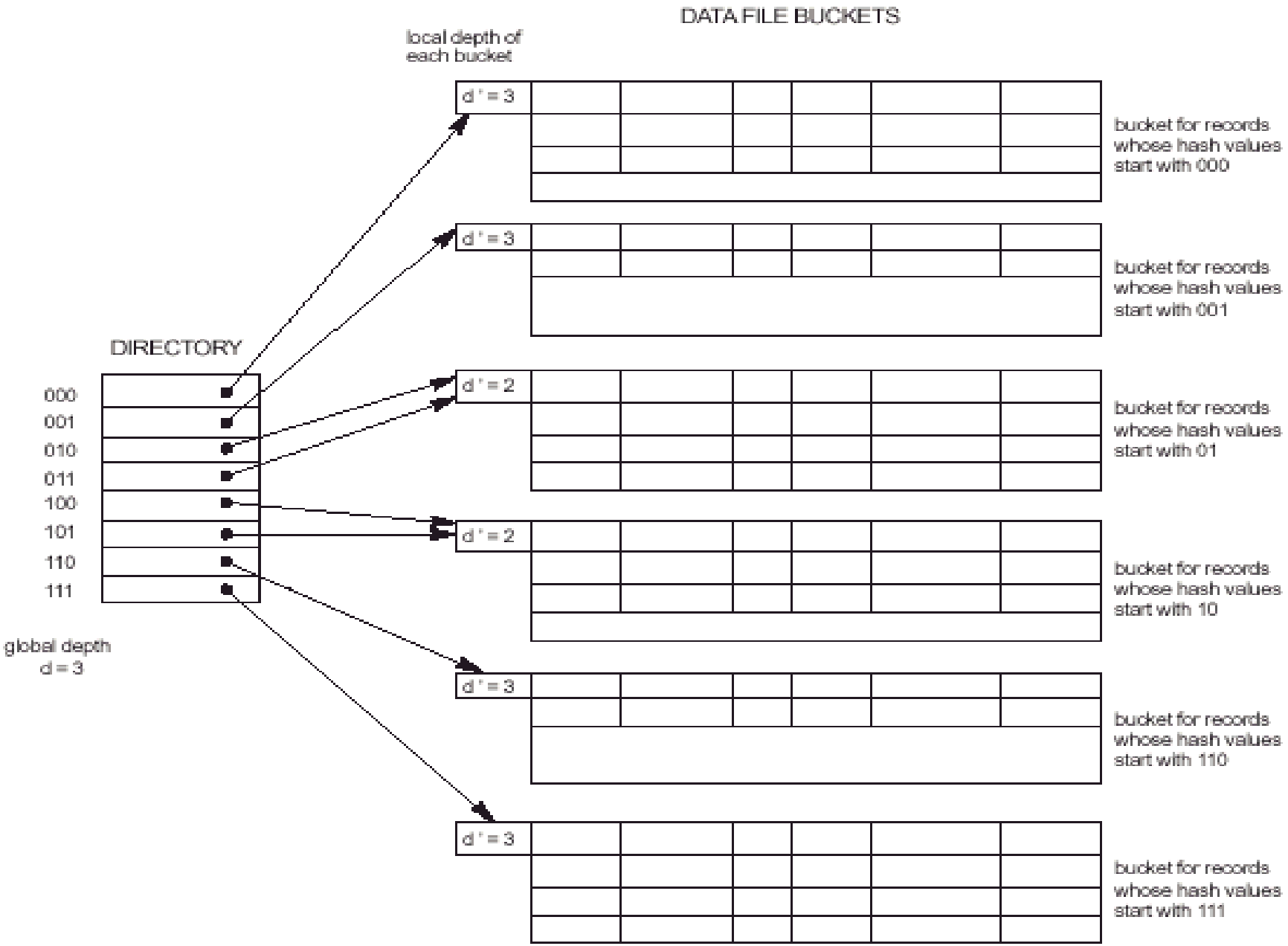



Fig 5.13 Structure of the extendible hashing scheme.

# Extendible Hashing(2)

- Incrementing *d* by one
  - Doubling the number of entries in the directory
  - When a bucket, whose *d'* is equal to *d*, overflows.
- Decrementing *d* by one
  - Halving the number of entries in the directory
  - When *d* > *d'* for all buckets after some deletions.
- Does not require an overflow area.
- Two block accesses for record retrieval.
  - One for directory
  - One for bucket

# Managing directory

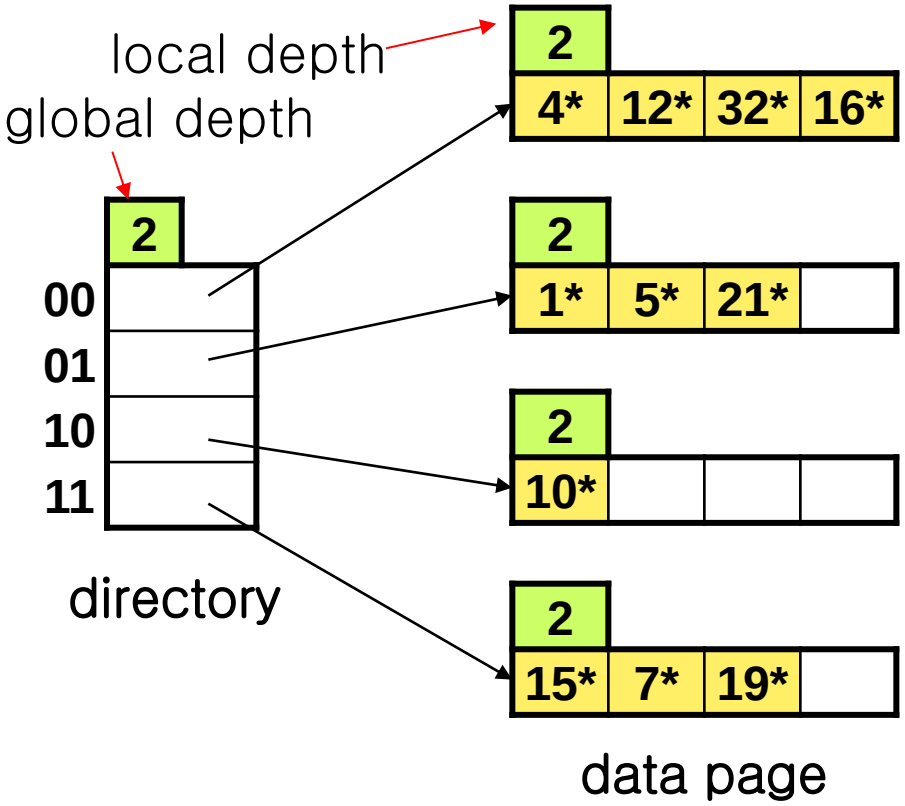
- When a bucket with *d'* = *d* overflows,
  - split the bucket
  - distribute records based on (*d*+*1*)th bit
  - double the directory
  - adjust the directory entries

- When d > d' for all the buckets,
  - halve the directory
  - adjust the directory entries

# Extendible hashing : 예제(1)

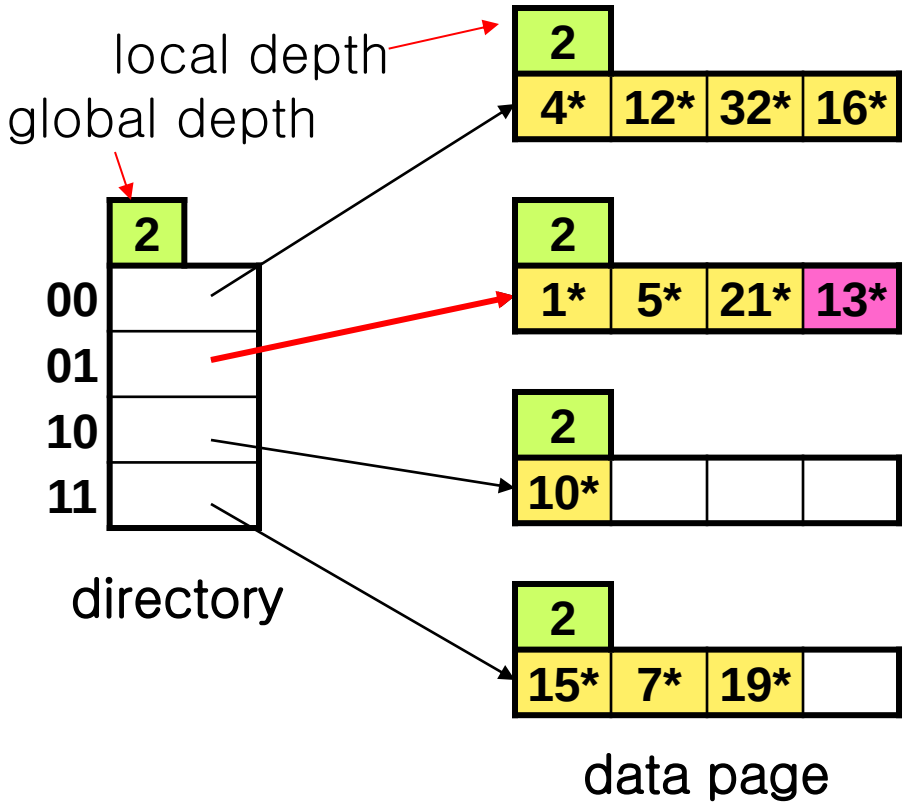
- Directory는 크기 4의 배열 (내용은 bucket에 대한 pointer)
- 각 bucket은 최대 4개의 data entry 저장
- $h(k)$ 값의 binary 수의 마지막 2 bit를 directory에 적용

(예) insert 13
13 mod 4 = 1(01)



# Extendible hashing : 예제(2)

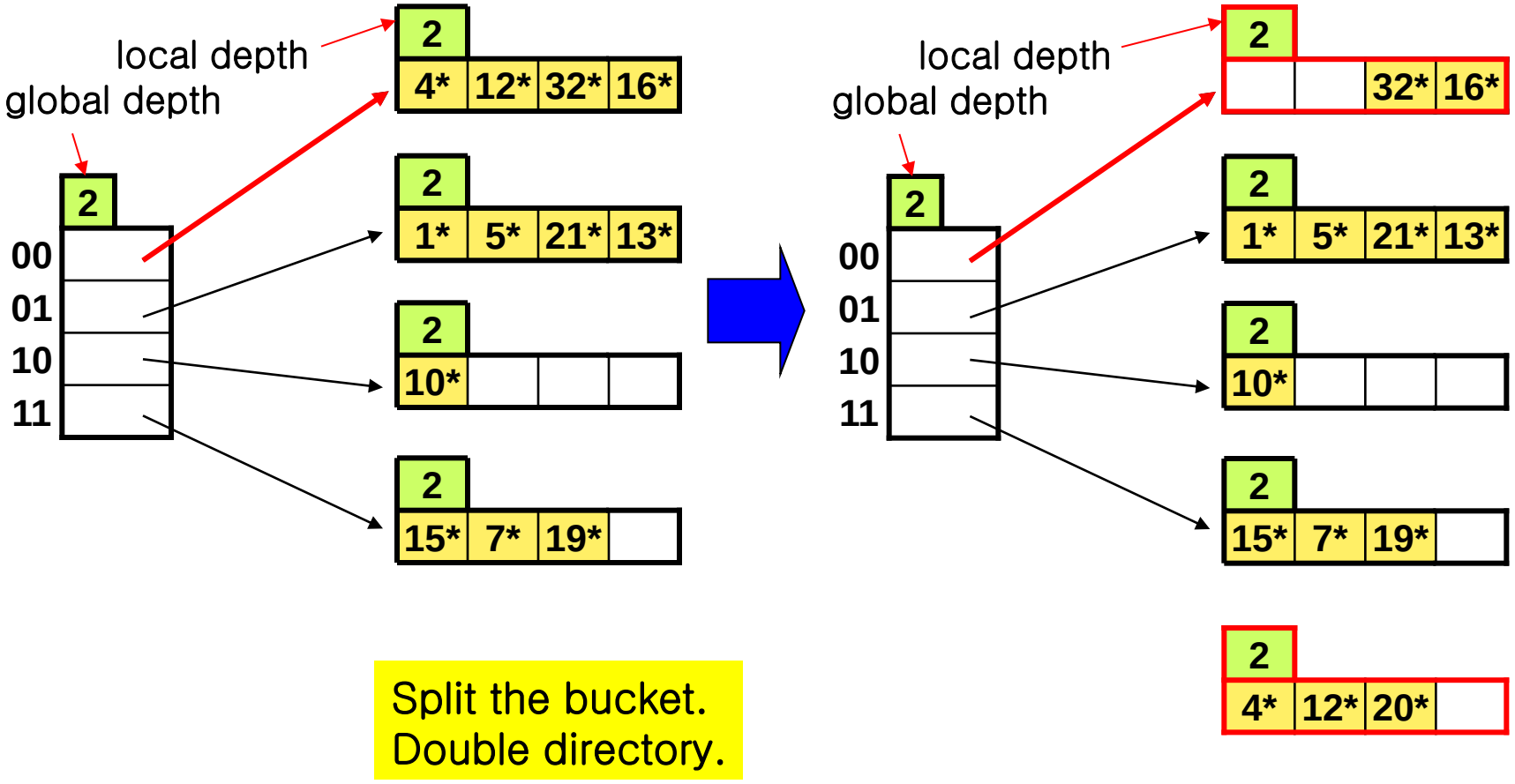
(예)
insert 13 (01)



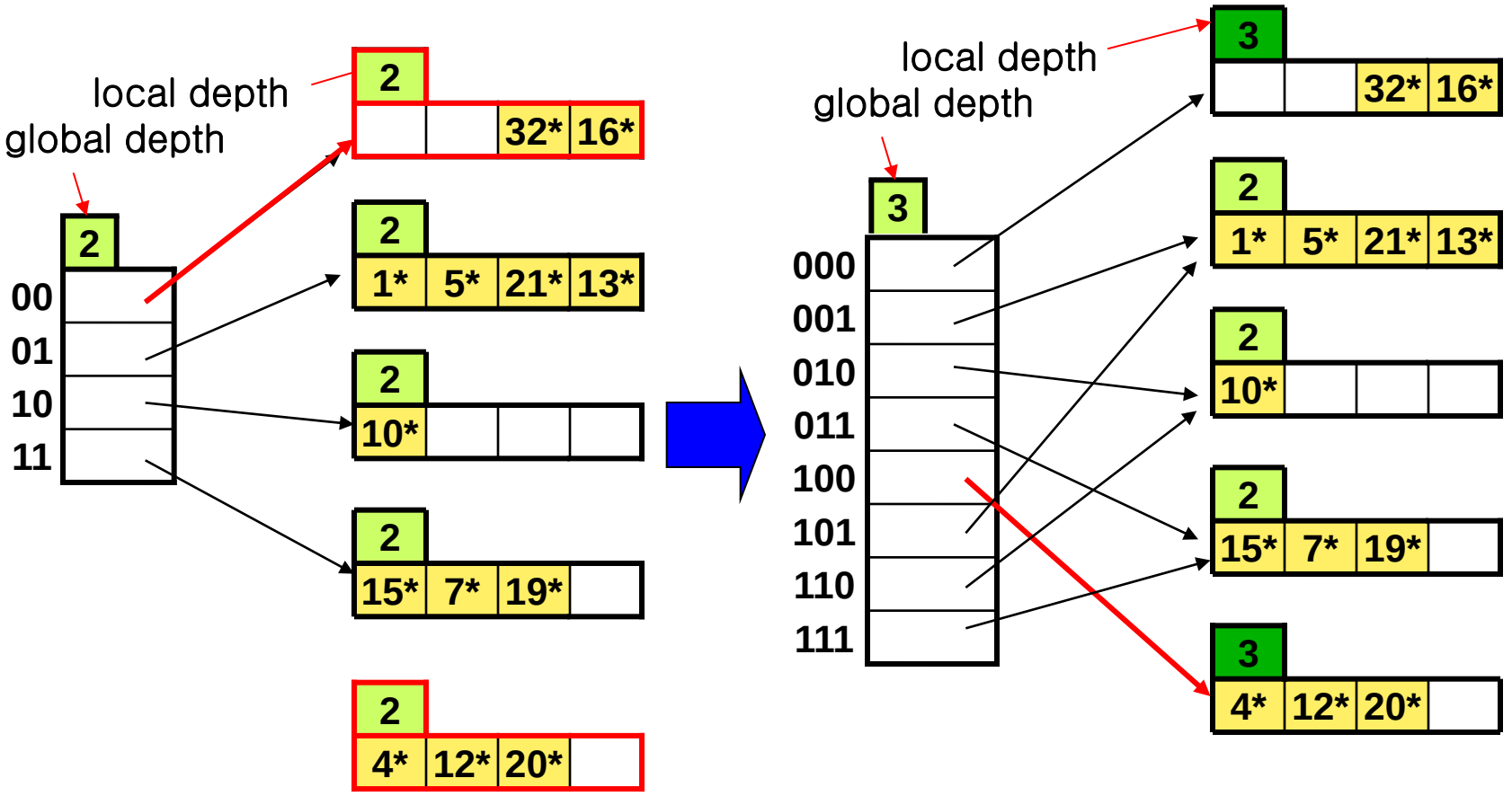
# Extendible hashing : 예제(3)

- full bucket에 data entry insert

(예) insert 20  (20 mod 4 = 0 (00))

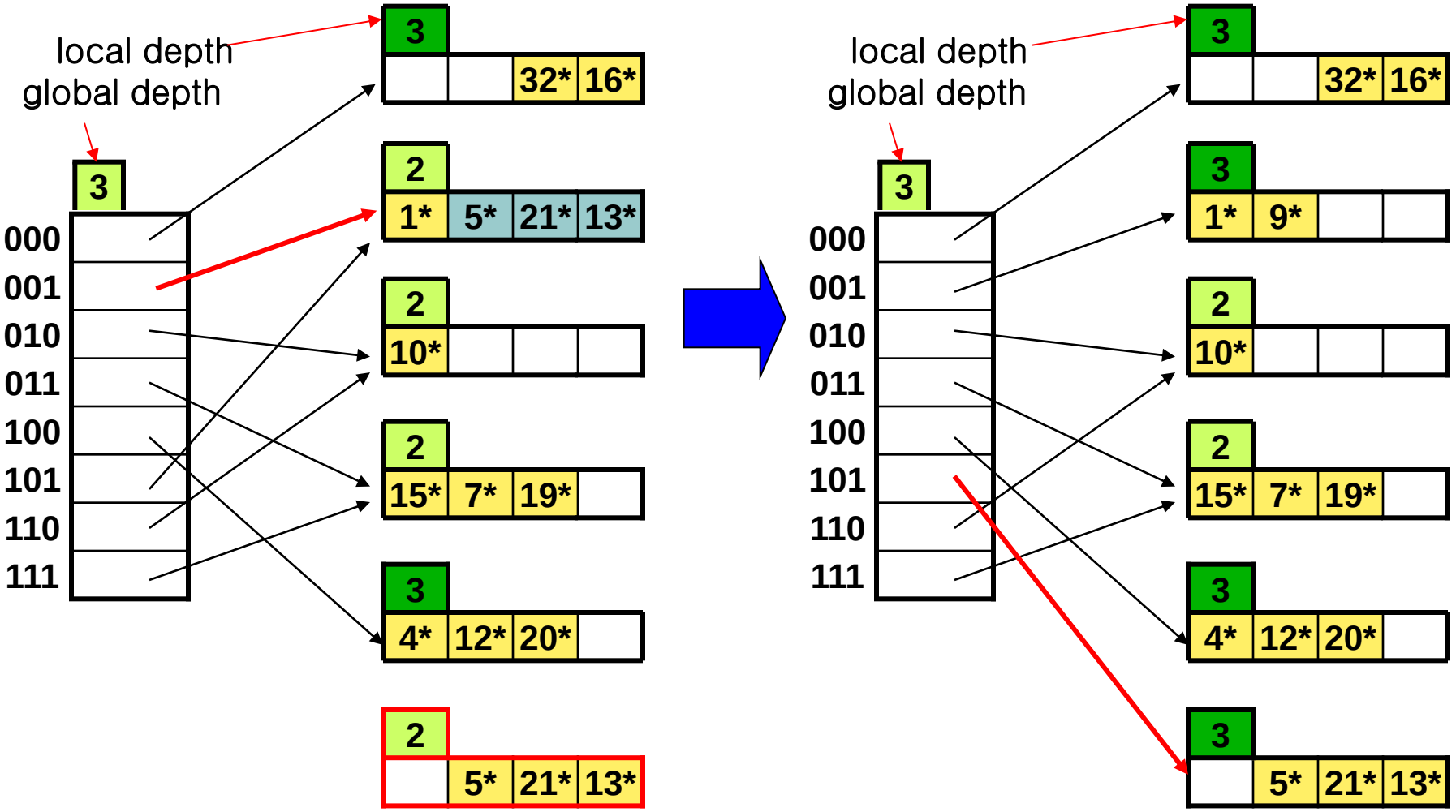


# Extendible hashing : 예제(4)



# Extendible hashing : 예제(5)

(예) insert 9 (9 mod 4 = 1 (01))



# Properties

- Advantages
  - Performance doesn’t degrade as the file grows
  - No additional bucket space allocated for future growth
  - Negligible directory space
  - Minor reorganization for splitting (redistribution occurs in the overflowed bucket only)

- Disadvantage
  - Two block accesses: for directory, and for the bucket
  - One block access in static hashing.

# Linear Hashing

- Dynamic expansion/shrinking of buckets without a directory
- Maintain overflow chains for each bucket
- For every overflow, buckets are split in the linear order.
- The overflowed bucket will eventually be split by the linear order → delayed split.
- Any records hashed to bucket *k* based on $h_i$ will hash to bucket *k* or bucket *k+M* based on $h_{i+1}$.

(Ex) $h_1(r) = r \bmod M$ / $h_2(r) = r \bmod 2M$ / $h_3(r) = r \bmod 4M$

# Operations

- When a collision occurs,
  - put the record into its overflow chain
  - split the bucket ***k (starting from 0) pointed by n*** into bucket *k* and bucket *k*+*M*
  - redistribute the records into the split buckets(*k*, *k*+*M*) using another hash function with $h_{i+1}$
- To retrieve a record with key *K*
  - apply hash function $h_i$
  - $h_i(K) < n$, apply $h_{i+1}(K)$
- When $n = M$
  - replace hash function and initialize $n$ to 0

# Search Algorithm for linear hashing

**if** $n = 0$

**then** $m \leftarrow h_j(k)$

**else begin**

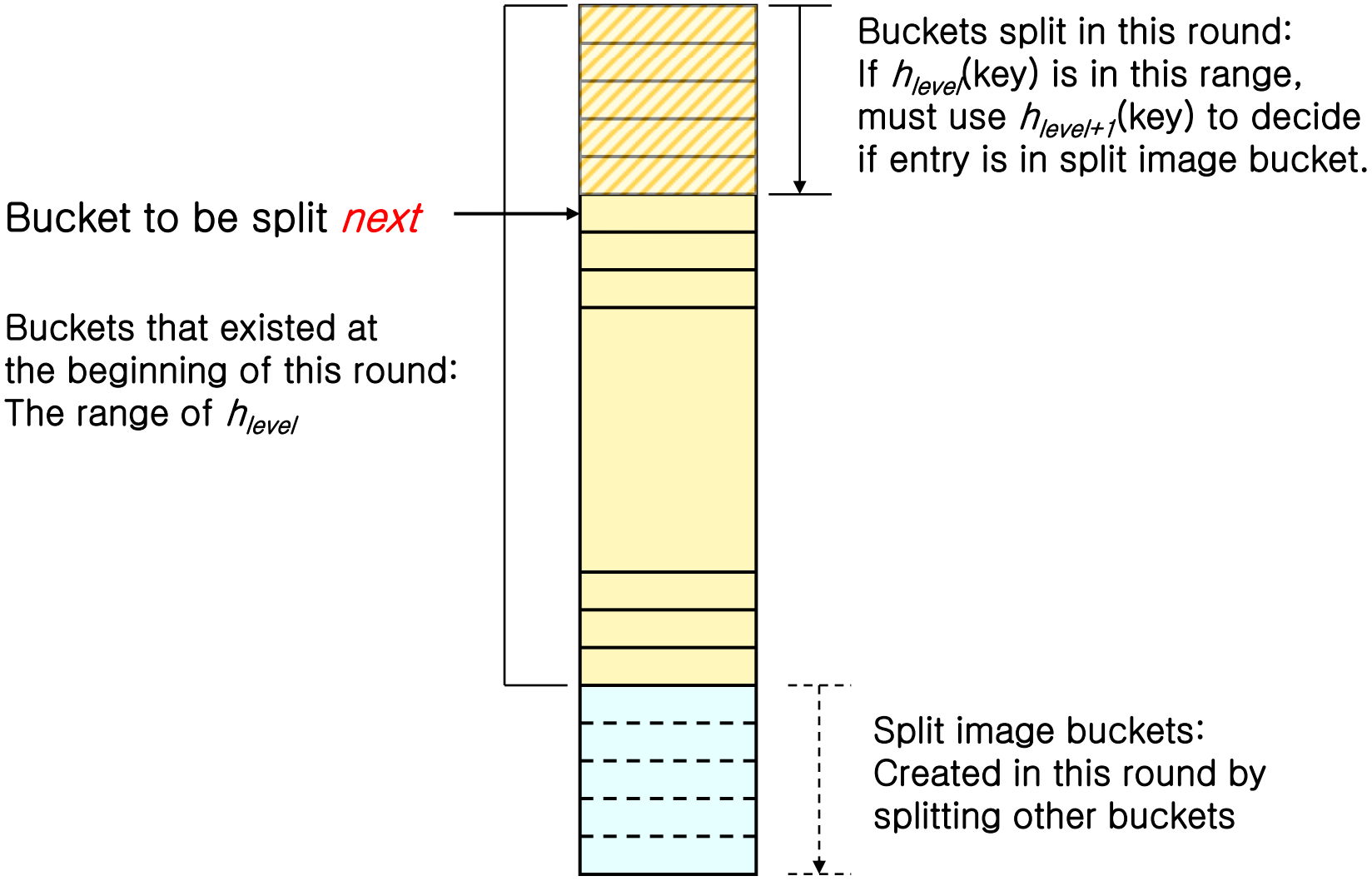
$m \leftarrow h_j(k)$

**if** $m < n$ **then** $m \leftarrow h_{j+1}(k)$

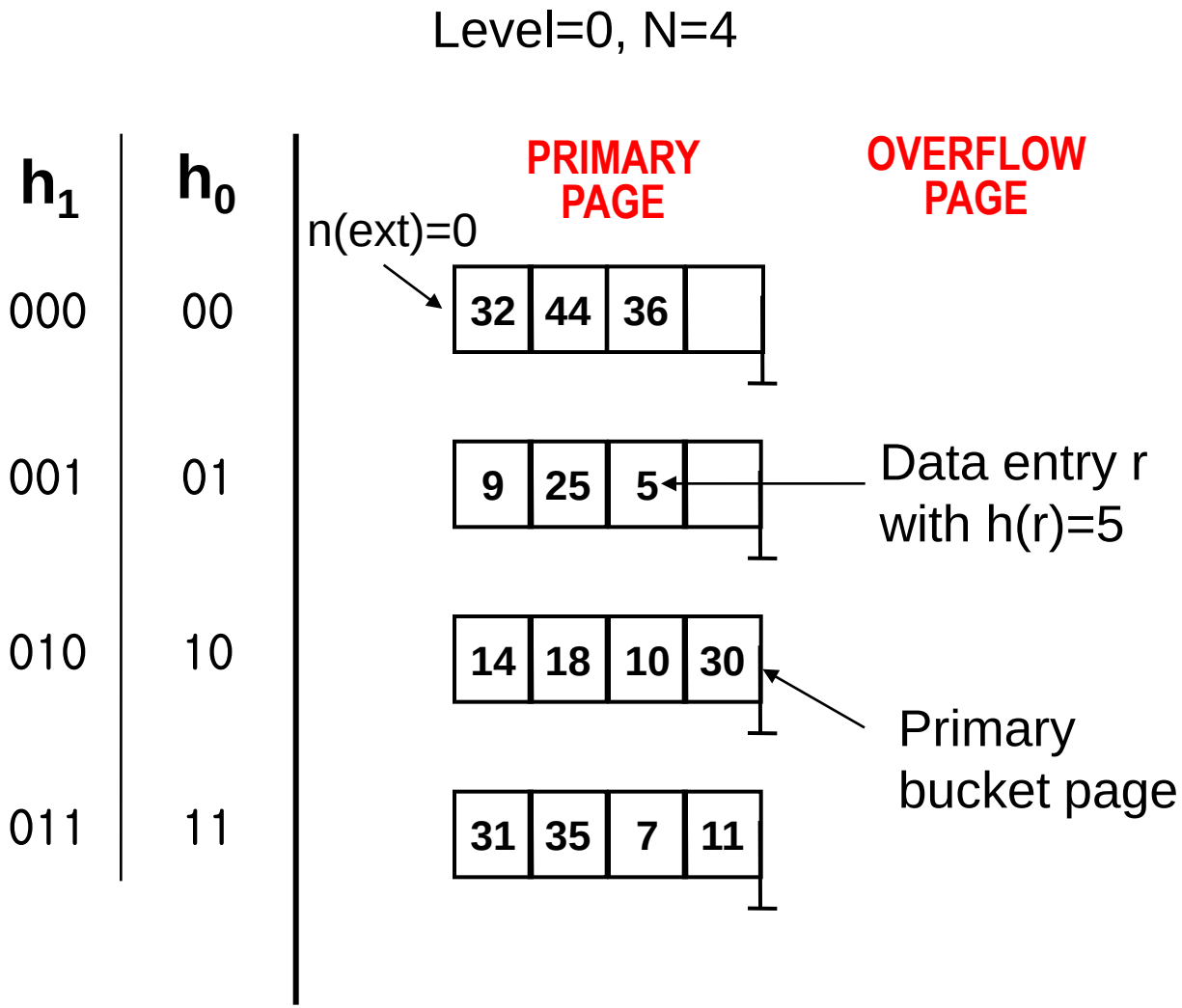
**end**

search the bucket whose hash value is $m$
( and its overflow if any)

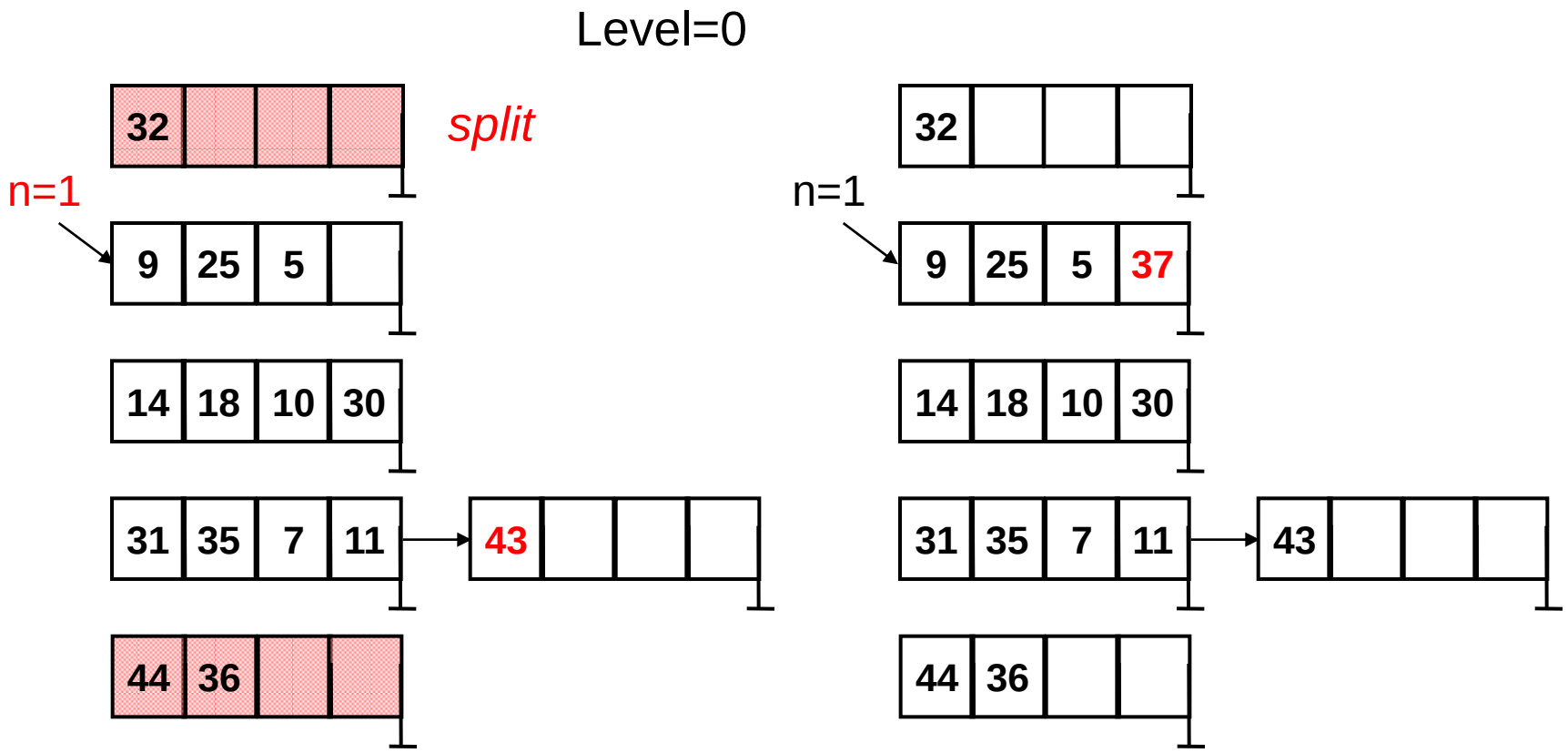
# Buckets during a Round



# Linear Hashing 예제(1)



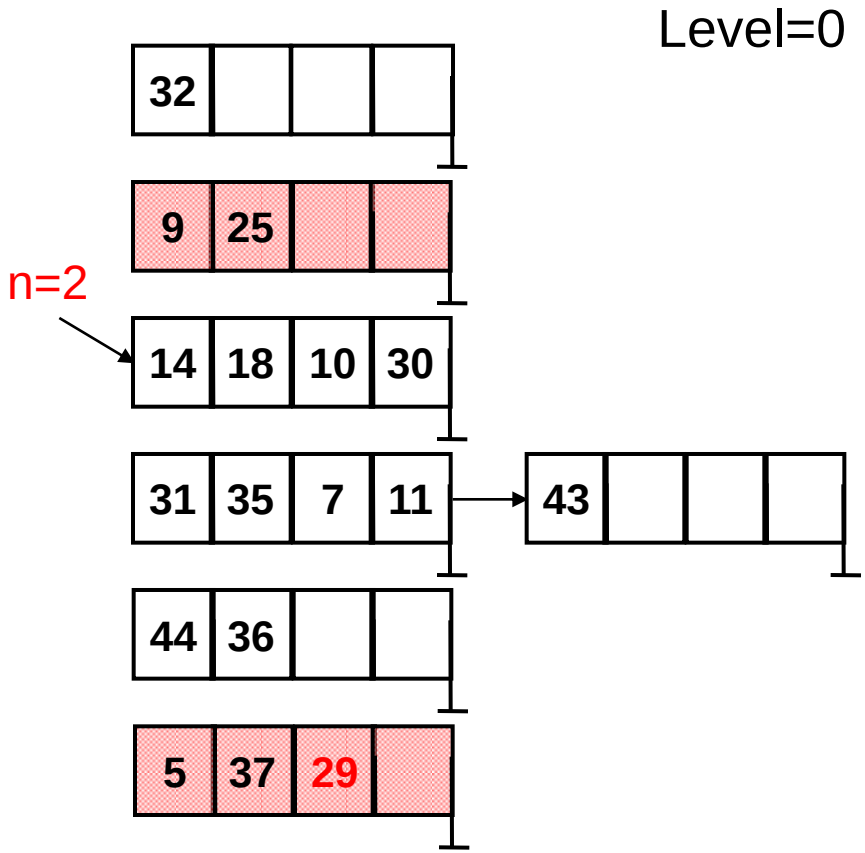
# Linear hashing 예제(2)



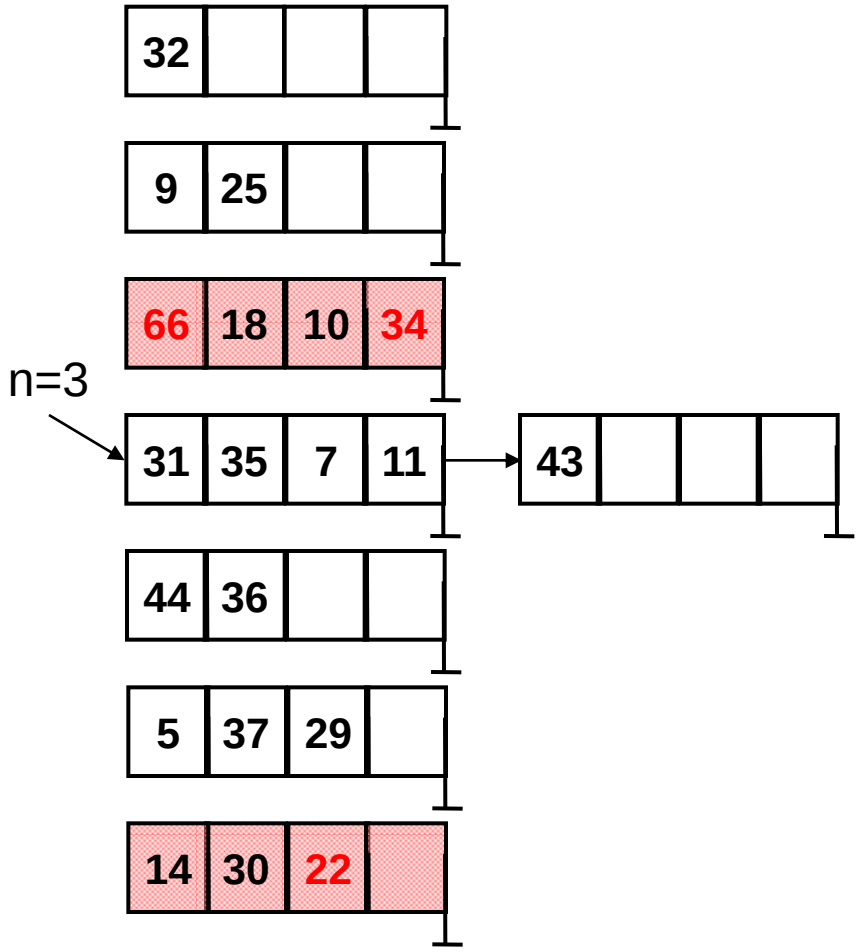

**(a) Insert a record with 43**

**(b) Insert a record with 37**

# Linear hashing 예제(3)

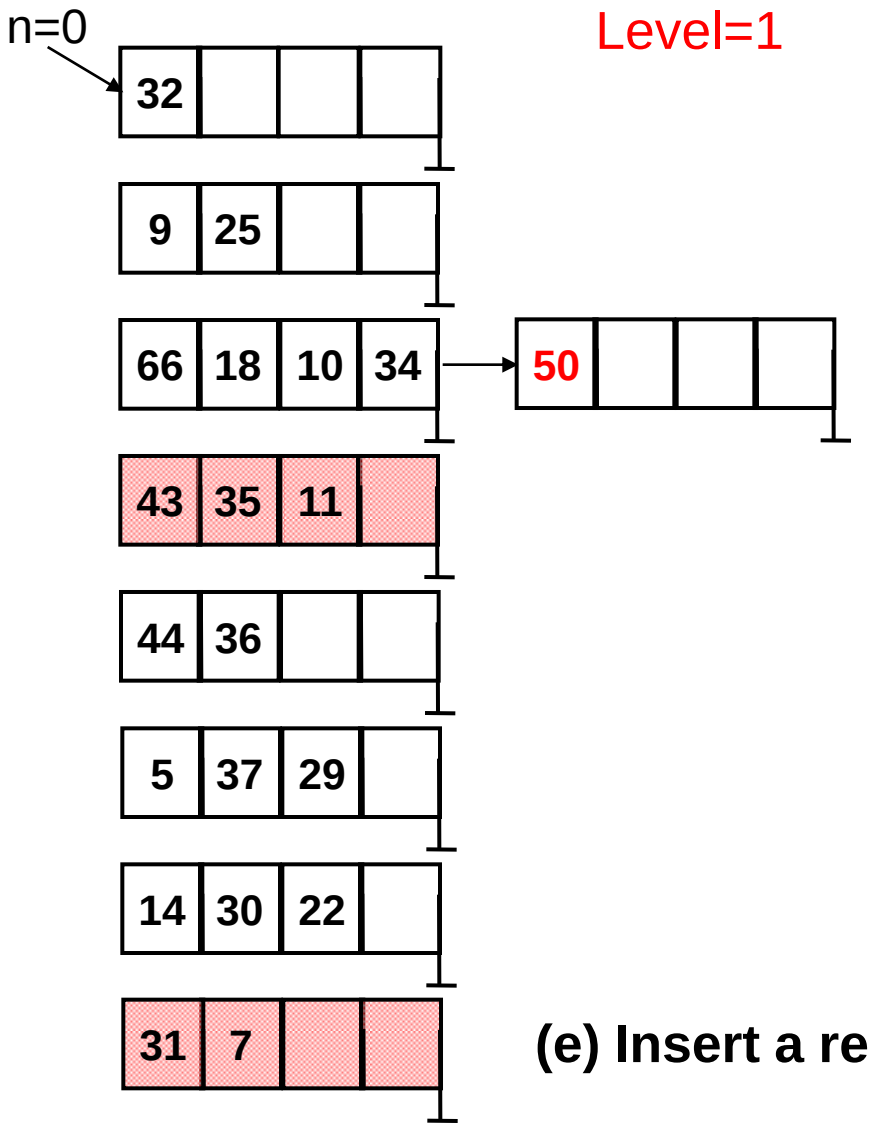



(c) Insert a record with 29




(d) Insert a record with 22,66,34

# Linear hashing 예제(4)




**(e) Insert a record with 50**